Herramientas básicas para el administrador

Josep Jorba Esteve

P07/M2103/02282



Índice

Introducción 5

1. Herramientas gráficas y líneas de comandos 7

2. Documentos de estándares 9

3. Documentación del sistema en línea 12

4. *Shells* y *Scripts* 14
4.1. *Shells* interactivos 15
4.2. *Shells* disponibles 18
4.3. Variables del sistema 21
4.4. Programación *scripts* en Bash 22
4.4.1. Variables en Bash 22
4.4.2. Comparaciones 23
4.4.3. Estructuras de control 24

5. Herramientas de gestión de paquetes 26
5.1. Paquete TGZ 27
5.2. Fedora/Red Hat: paquetes RPM 29
5.3. Debian: paquetes DEB 33

6. Herramientas genéricas de administración 38

7. Otras herramientas 40

Actividades 41

Otras fuentes de referencia e información 41

Introducción

El administrador de sistemas GNU/Linux tiene que enfrentarse diariamente a una gran cantidad de tareas. En general, en la filosofía UNIX no suele haber una única herramienta para cada tarea o una sola manera de hacer las cosas. Lo común es que los sistemas UNIX proporcionen una gran cantidad de herramientas más o menos simples para afrontar las diferentes tareas.

Será la combinación de las herramientas básicas, cada una con una tarea muy definida, la que nos dará la posibilidad de solucionar un problema o tarea de administración.

Nota

GNU/Linux posee un conjunto muy amplio de herramientas con funcionalidades básicas, cuya potencia está en su combinación.

En esta unidad veremos diferentes grupos de herramientas, identificaremos algunas de sus funciones básicas, y veremos algunos ejemplos de sus usos. Comenzaremos por examinar algunos estándares del mundo GNU/Linux, que nos permitirán hallar algunas de las características básicas que esperamos de cualquier distribución de GNU/Linux. Estos estándares, como el LSB (o *Linux standard base*) [Linc] y el FHS (*filesystem hierarchy standard*) [Linb], nos hablan de herramientas que esperamos encontrar disponibles, de una estructura común para el sistema de ficheros, así como de diversas normas que tienen que cumplirse para que una distribución sea considerada un sistema GNU/Linux y mantenga reglas comunes para la compatibilidad entre ellos.

En la automatización de tareas de administración suelen utilizarse comandos agrupados en *shell scripts* (también llamados guiones de comandos), mediante lenguaje interpretados por el *shell* (intérprete de comandos) del sistema. En la programación de estos *shell scripts* se nos permite unir los comandos del sistema con estructuras de control de flujo, y así disponer de un entorno de prototipo rápido de herramientas para la automatización de tareas.

Otro esquema habitual es la utilización de herramientas de compilación y depuración de lenguajes de alto nivel (como por ejemplo C). En general, serán utilizadas por el administrador para generar nuevos desarrollos de aplicaciones o herramientas, o para incorporar al sistema aplicaciones que vengan como código fuente y tengan que adaptarse y compilarse.

También analizaremos el uso de algunas herramientas gráficas con respecto a las habituales de la línea de comandos. Estas herramientas suelen facilitar las tareas al administrador, pero su uso es limitado, ya que dependen fuertemente de la distribución de GNU/Linux, o incluso de cada versión. Aun así, hay algunas herramientas útiles que son exportables entre distribuciones.

Por último, analizaremos un grupo de herramientas imprescindibles para mantener el sistema actualizado, las herramientas de gestión de paquetes. El software servido en la distribución GNU/Linux, o incorporado posteriormente, se suele ofrecer en unidades denominadas paquetes, que incluyen los archivos de un determinado software, más pasos diversos necesarios para la preparación de la instalación, la configuración posterior, o, si es el caso, la actualización o desinstalación de un determinado software. Y cada distribución suele aportar software de gestión para mantener las listas de paquetes instalados o por instalar, así como el control de las versiones existentes o posibilidades diversas de actualización por medio de diferentes fuentes origen.

1. Herramientas gráficas y líneas de comandos

Existe un gran número de herramientas, de las que examinamos una pequeña porción en éste y los siguientes módulos, que como herramientas de administración son proporcionadas por terceros de forma independiente a la distribución o por el mismo distribuidor del sistema GNU/Linux.

Estas herramientas pueden cubrir más o menos aspectos de la administración de una tarea concreta, y presentarse con múltiples interfaces diferentes: ya sean herramientas de línea de comandos con múltiples opciones y/o ficheros de configuración asociados, o herramientas textuales con algún tipo de menús elaborados; o bien herramientas gráficas con interfaces más adecuadas para el manejo de información, o asistentes que automaticen las tareas, o bien interfaces web de administración.

Todo esto nos ofrece un gran número de posibilidades de cara a la administración, pero siempre tenemos que valorar su facilidad de uso junto con las prestaciones, y los conocimientos que posea el administrador que se dedica a estas tareas.

Las tareas habituales del administrador GNU/Linux pueden pasar por trabajar con diferentes distribuciones (por ejemplo, las que comentaremos Fedora [Fed] o Debian [Debb], o cualquier otra), o incluso trabajar con variantes comerciales de otros UNIX. Esto conlleva que tengamos que establecer una cierta manera de trabajar que nos permita hacer de forma uniforme las tareas en los diferentes sistemas.

Por esta razón, a lo largo de los diferentes módulos intentaremos destacar todos aquellos aspectos más comunes, y las técnicas de administración serán realizadas en su mayor parte a bajo nivel, mediante una línea de comandos y/o con edición de ficheros de configuración asociados.

Cualquiera de las distribuciones de GNU/Linux suele aportar herramientas del tipo línea de comandos, textual o, en particular, gráficas, que complementan las anteriores y simplifican en mayor o menor medida la administración de las tareas [Sm02]. Pero hay que tener en cuenta varias cosas:

a) Estas herramientas son una interfaz más o menos elaborada de las herramientas básicas de línea de comandos y los correspondientes ficheros de configuración.

b) Normalmente no ofrecen todas las prestaciones o configuraciones que pueden realizarse a bajo nivel.

c) Los errores pueden no gestionarse bien, o simplemente proporcionar mensajes tipo “la tarea no se ha podido realizar”.

d) El uso de estas herramientas oculta, a veces completamente, el funcionamiento interno del servicio o tarea. Comprender bien el funcionamiento interno es un conocimiento básico para el administrador, y más si tiene que desarrollar tareas de corrección de errores u optimización de servicios.

e) Estas herramientas son útiles en la mejora de la producción, una vez que el administrador tiene los conocimientos adecuados, ya que puede manejar con ellas de forma más eficaz las tareas rutinarias y automatizarlas.

f) O también el caso contrario, la tarea puede ser tan compleja, o necesitar tantos parámetros, o generar tantos datos, que se vuelve imposible controlarla de forma manual. En estos casos, las herramientas de alto nivel pueden ser muy útiles y volver practicables algunas tareas que de otra manera son difíciles de controlar. Por ejemplo, dentro de esta categoría entrarían las herramientas de visualización, monitorización, y resumen de actividades o servicios complejos.

g) En la automatización de tareas, estas herramientas (de más alto nivel) pueden no ser las adecuadas: pueden no haber estado pensadas para los pasos que hay que realizar, o bien hacerlo de una forma no eficaz. Por ejemplo, un caso concreto puede ser la creación de usuarios, una herramienta visual puede ser muy atrayente, por la forma de introducir los datos, pero ¿qué sucede cuando en lugar de introducir uno o unos cuantos usuarios queremos introducir una lista de decenas o centenares de éstos?, la herramienta, si no está preparada, se vuelve totalmente ineficiente.

h) Por ultimo, los administradores suelen querer personalizar sus tareas utilizando las herramientas que consideran más cómodas y fáciles de adaptar. En este aspecto, suele ser habitual la utilización de las herramientas básicas de bajo nivel, y la utilización de *shell scripts* (veremos los fundamentos en esta unidad) para combinarlas de modo que formen una tarea.

Tenemos que saber valorar estas herramientas extra según la valía que tengan para nuestras tareas.

Podemos dar a estas herramientas un uso casual (o cotidiano), si tenemos los conocimientos suficientes para tratar los errores que puedan producirse, o bien con objetivo de facilitar algún proceso para el que haya sido pensada la herramienta, pero siempre controlando las tareas que implementamos y el conocimiento técnico subyacente.

2. Documentos de estándares

Los estándares, ya sean genéricos del mundo UNIX o particulares de GNU/Linux, nos permiten seguir unos criterios básicos, por los que nos guiamos en el momento de aprender o realizar una tarea, y que nos proporcionan información básica para comenzar nuestro trabajo.

En GNU/Linux podemos encontrarnos con estándares como el FHS (*filesystem hierarchy standard*) [Linb], que nos explica qué podemos encontrarnos (o dónde buscarlo) en la estructura del sistema de ficheros de nuestro sistema. O el LSB (*Linux standard base)*, que nos comenta diferentes componentes que solemos encontrar en los sistemas [Linc].

Nota

Ver FHS en:
www.pathname.com/fhs

En el estándar **FHS** (*filesystem hierchachy standard*) se describen la estructura de árbol del sistema de ficheros principal (/), donde se especifica la estructura de los directorios y los principales ficheros que contendrán. Este estándar es usado en mayor o menor medida también para los UNIX comerciales, en los cuales al principio hubo muchas diferencias que hicieron que cada fabricante cambiara la estructura a su gusto. El estándar pensado en origen para GNU/Linux se hizo para normalizar esta situación y evitar cambios drásticos. Aun así, el estándar es seguido con diferentes grados, la mayoría de distribuciones siguen en un alto porcentaje el FHS, realizando cambios menores o aportando ficheros o directorios que no existían en el estándar.

Nota

El estándar FHS es una herramienta básica para el conocimiento de una distribución, que nos permite conocer la estructura y funcionalidad del sistema de archivos principal del sistema.

Un esquema básico de directorios podría ser:

- /bin: utilidades de base del sistema, normalmente programas empleados por los usuarios, ya sean desde los comandos básicos del sistema (como /bin/ls, listar directorio), pasando por los *shells* (/bin/bash), etc.

- /boot: archivos necesarios durante el arranque del sistema, por ejemplo la imagen del *kernel* Linux, en /boot/vmlinuz.

- /dev: aquí encontramos ficheros especiales que representan los dispositivos posibles en el sistema, el acceso a los periféricos en sistemas UNIX se hace como si fueran ficheros. Podemos encontrar ficheros como /dev/console, /dev/modem, /dev/mouse, /dev/cdrom, /dev/floppy... que suelen ser enlaces a dispositivos más específicos del tipo de controlador o interfaz que utilizan los dispositivos: /dev/mouse, enlazado a /dev/psaux, representado un ratón de tipo PS2; o /dev/cdrom a /dev/hdc, un CD-ROM que es un dispositivo del segundo conector IDE y máster. Aquí encontramos los dispositivos IDE como /dev/hdx, los scsi /dev/sdx... con x variando según el

número de dispositivo. Aquí cabe comentar que en su inicio este directorio era estático, con los ficheros predefinidos, y/o configurados en determinados momentos, hiy en dia se utilizan técnicas tecnologías dinámicas (como *hotplug* u *udev*), que permiten detectar dispositivos y crear los archivos /dev dinámicamente bien al inicio del sistema, o durante la ejecución, con la inserción de dispositivos removibles.

- /etc: ficheros de configuración. La mayoría de tareas de administración necesitarán examinar o modificar los ficheros contenidos en este directorio. Por ejemplo: /etc/passwd contiene parte de la información de las cuentas de los usuarios del sistema.

- /home: contiene las cuentas de los usuarios, es decir, los directorios personales de cada usuario.

- /lib: las bibliotecas del sistema, compartidas por los programas de usuario, ya sean estáticas (extensión .a) o dinámicas (extensión .so). Por ejemplo, la biblioteca C estándar, en ficheros libc.so o libc.a. También en particular suelen encontrarse los módulos dinámicos del *kernel* Linux, en /lib/modules.

- /mnt: punto para montar (comando mount) sistemas de ficheros de forma temporal; por ejemplo: /mnt/cdrom, para montar un disco en el lector de CD-ROM momentáneamente.

- /media: para punto de montaje habitual de dispositivos extraíbles.

- /opt: suele colocarse el software añadido al sistema posterior a la instalación; otra instalación válida es en /usr/local.

- /sbin: utilidades de base del sistema. Suelen ser comandos reservados al administrador (*root*). Por ejemplo: /sbin/fsck para verificar el estado de los sistemas de ficheros.

- /tmp: ficheros temporales de las aplicaciones o del propio sistema. Aunque son para la ejecución temporal, entre dos ejecuciones la aplicación/servicio no puede asumir que va a encontrar los anteriores ficheros.

- /usr: diferentes elementos instalados en el sistema. Algún software de sistema más completo se instala aquí, además de complementos multimedia (iconos, imágenes, sonidos, por ejemplo en: /usr/share) y la documentación del sistema (/usr/share/doc). También en /usr/local se suele utilizar para instalar software.

- /var: ficheros de registro de sesión o de estado (ficheros de tipo log) y/o errores del propio sistema y de diversos servicios, tanto locales como de

red. Por ejemplo, ficheros de sesión en /var/log, contenido de los correos en /var/spool/mail, o trabajos de impresión en /var/spool/lpd.

Éstos son algunos de los directorios definidos en el FHS para el sistema raíz, luego se especifican por ejemplo algunas subdivisiones, como el contenido de los /usr y /var, y los ficheros de datos y/o ejecutables típicos que se esperan encontrar como mínimo en los directorios (ver las referencias a los documentos FHS).

Respecto a las distribuciones, Fedora/Red Hat sigue el estándar FHS muy de cerca. Sólo presenta algunos cambios en los archivos presentes en /usr, /var. En /etc suele existir un directorio por componente configurable, y en /opt, /usr/local no suele existir software instalado, a no ser que el usuario lo instale. Debian, por su parte, sigue el estándar, aunque añade algunos directorios de configuración especiales en /etc.

Nota

http://www.linuxbase.org/spec/refsp ecs/LSB_1.3.0/gLSB/gLSB/rstandard s.html

Otro estándar en proceso es el LSB (*Linux standard base*) [Linc]. La idea de éste es definir unos niveles de compatibilidad entre las aplicaciones, bibliotecas y utilidades, de manera que sea posible la portabilidad de las aplicaciones entre distribuciones sin demasiados problemas. Además del estándar, proporcionan conjuntos de prueba (tests) para verificar el nivel de compatibilidad. LSB en sí mismo es un recopilatorio de varios estándares aplicados a GNU/Linux.

Nota

http://www.linuxbase.org/spec/

3. Documentación del sistema en línea

Uno de los aspectos más importantes para nuestras tareas de administración será disponer de la documentación correcta para nuestro sistema y el software instalado. Hay muchas fuentes de información, pero destacaremos las siguientes:

a) man es la ayuda por excelencia. Nos permite consultar el manual de GNU/ Linux, que está agrupado en varias secciones, correspondientes a comandos administración, formatos de ficheros, comandos de usuario, llamadas de lenguaje C, etc. Normalmente, para obtener la ayuda asociada, tendremos suficiente con:

```
man comando
```

Cada página describiría el comando junto con sus opciones y, normalmente, algunos ejemplos de utilización. A veces, puede haber más de una entrada en el manual. Por ejemplo, puede que haya una llamada C con igual nombre que un comando; en este caso, hay que especificar qué sección quiere mirarse:

```
man n comando
```

siendo *n* el número de sección.

Existen también unas cuantas herramientas de exploración de los manuales, por ejemplo xman y tkman, que mediante interfaz gráfica facilitan el examen de las diferentes secciones, así como índices de los comandos. Otro comando interesante es *apropos* palabra, que nos permitirá localizar páginas man que hablen de un tema determinado (asociado con la palabra buscada).

b) info es otro sistema de ayuda habitual. Es un programa desarrollado por GNU para la documentación de muchas de sus herramientas. Es básicamente una herramienta textual en la que los capítulos y páginas se pueden recorrer por medio de un sistema de navegación simple (basado en teclado).

c) Documentación de las aplicaciones: además de ciertas páginas man, es habitual incluir en las aplicaciones documentación extra, ya sea en forma de manuales o tutoriales, o simples guías de usuario. Normalmente, estos componentes de documentación se instalan en el directorio /usr/share/doc (O /usr/doc dependiendo de la distribución), donde normalmente se crea un directorio por paquete de aplicación (normalmente la aplicación puede disponer de paquete de documentación por separado).

d) Sistemas propios de las distribuciones. Red Hat suele venir con unos CD de manuales de consulta que son instalables en el sistema y tienen formatos HTML o PDF. Fedora dispone de un proyecto de documentación en su web. Debian trae los manuales como un paquete de software más y suelen instalarse en /usr/doc. Por otra parte, dispone de herramientas que clasifican la documentación presente en el sistema, y la organizan por menús para su visualización, como dwww o dhelp, las cuales presentan interfaces web para examinar la documentación del sistema.

e) Por último, los escritorios X, como Gnome y KDE, normalmente también traen sistemas de documentación propios con su documentación y manuales, así como información para desarrolladores, ya sea en forma de ayudas gráficas en sus aplicaciones, o en aplicaciones propias que recopilan las ayudas (por ejemplo devhelp en Gnome).

4. *Shells* y *Scripts*

El término genérico *shell* se utiliza para denominar un programa que sirve de interfaz entre el usuario y el núcleo (*kernel*) del sistema GNU/Linux. En este apartado nos centraremos en los *shell*s interactivos de texto, que serán los que nos encontraremos como usuarios una vez estemos validados y dentro del sistema.

El *shell*, como programa, es una utilidad de sistema, que permite a los usuarios interactuar con el *kernel* por interpretación de comandos que el mismo usuario introduce en la línea de comandos o en los ficheros de tipo *shell script*.

El *shell* es lo que los usuarios ven del sistema. El resto del sistema operativo permanece esencialmente oculto a sus ojos. El *shell* está escrito de la misma forma que un proceso (programa) de usuario; no está integrado en el *kernel*, sino que se ejecuta como un programa más del usuario.

Cuando nuestro sistema GNU/Linux arranca, suele presentar a los usuarios una interfaz de cara determinada; esta interfaz puede ser tanto de texto, como gráfica. Dependiendo de los modos (o niveles) de arranque del sistema, ya sea con los diferentes modos de consola de texto o con modos donde directamente tengamos arranque gráfico en X Window.

En los modos de arranque gráfico, la interfaz está compuesta por algún administrador de acceso que gestiona el proceso de *login* del usuario desde una "carátula" gráfica, en la que se le pide la información de entrada correspondiente: su identificador como usuario y su palabra de paso (o *passsword*). En GNU/Linux suelen ser habituales los gestores de acceso: xdm (propio de X Window), gdm (Gnome) y kdm (KDE), así como algún otro asociado a diferentes gestores de ventanas *(window managers*). Una vez validado nuestro acceso, nos encontraremos dentro de la interfaz gráfica de X Window con algún gestor de ventanas, como Gnome o KDE. Para interactuar desde un *shell* interactivo, sólo tendremos que abrir alguno de los programas de emulación de terminal disponibles.

Si nuestro acceso es por modo consola (en texto), una vez validados obtendremos el acceso directo al *shell* interactivo.

Otro caso de obtención de un *shell* interactivo es el acceso remoto a la máquina, ya sea vía cualquiera de las posibilidades de texto como telnet, r*login*, ssh, o gráficas como los emuladores X Window.

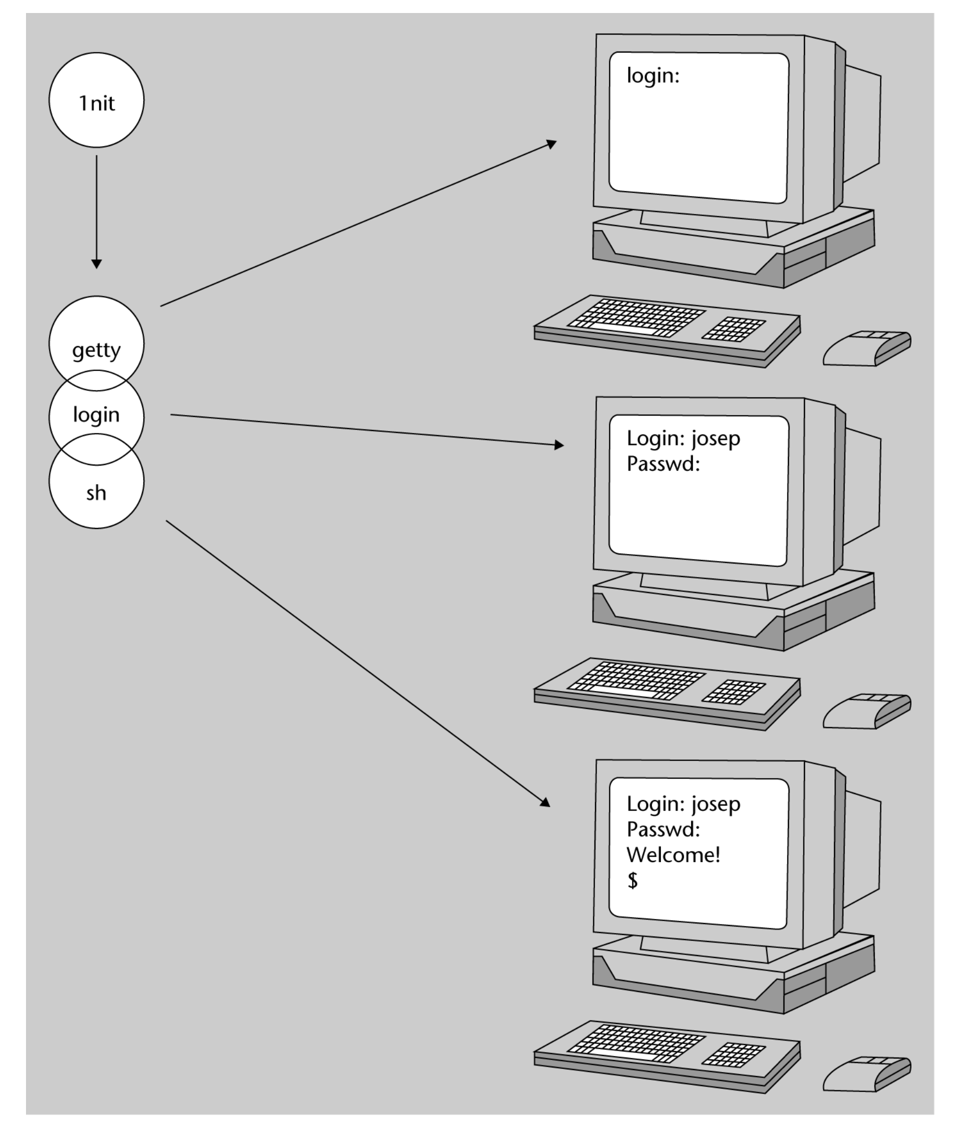


Figura 1. Ejemplo de arranque *shell* textual y procesos de sistema involucrados [Oke]

4.1. *Shells* interactivos

Una vez iniciado el *shell* interactivo [Qui01], se muestra un prompt de cara al usuario, indicándole que puede introducir una línea de comando. Tras la introducción, el *shell* asume la responsabilidad de validarla y poner los procesos necesarios en ejecución, mediante una serie de fases:

- Leer e interpretar la línea de comandos.
- Evaluar los caracteres "comodín" como $ * ? u otros.
- Gestionar las redirecciones de E/S necesarias, los pipes y los procesos en segundo plano (*background*) necesarios (&).
- Manejar señales.
- Preparar la ejecución de los programas.

Normalmente, las líneas de comandos podrán ser ejecuciones de comandos del sistema, comandos propios del *shell* interactivo, puesta en marcha de aplicaciones o *shell scripts*.

Los *shell scripts* son ficheros de texto que contienen secuencias de comandos de sistema, más una serie de comandos propios internos del *shell* interactivo, más las estructuras de control necesarias para procesar el flujo del programa (tipo *while, for*, etc.).

Los ficheros *script* son directamente ejecutables por el sistema bajo el nombre que se haya dado al fichero. Para ejecutarlos, se invoca el *shell* junto con el nombre del fichero, o bien se dan permisos de ejecución al *shell script*.

En cierta manera, podemos ver el *shell script* como código de un lenguaje interpretado que se ejecuta sobre el *shell* interactivo correspondiente. Para el administrador, los *shell scripts* son muy importantes básicamente por dos razones:

1) La configuración del sistema y de la mayoría de los servicios proporcionados se hacen mediante herramientas proporcionadas en forma de *shell scripts*.

2) La principal forma de automatizar procesos de administración es mediante la creación de *shell scripts* por parte del administrador.

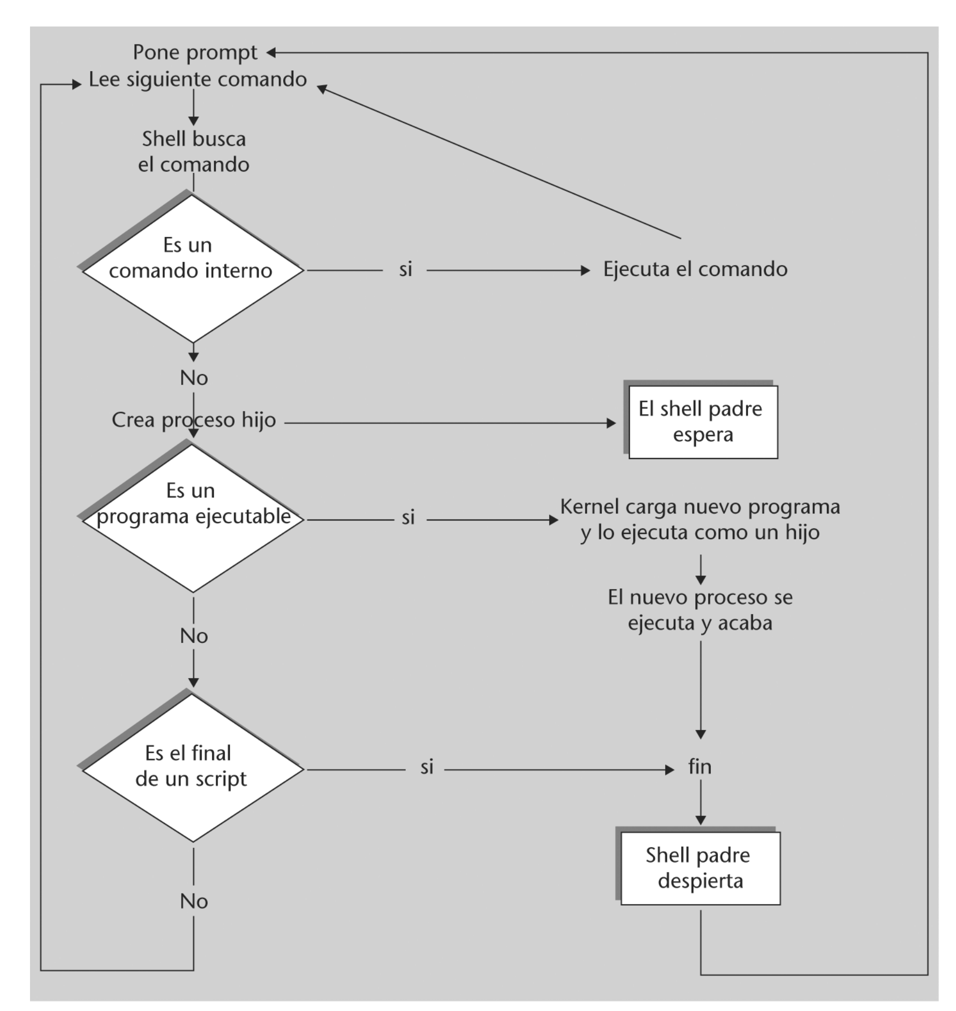


Figura 2. Flujo de control básico de un *shell*

Todos los programas invocados mediante un *shell* poseen tres ficheros predefinidos, especificados por los correspondientes de*script*ores de ficheros (*file handles*). Por defecto, estos ficheros son:

1) *standard input* (entrada estándar): normalmente asignada al teclado del terminal (consola); usa el de*script*or número 0 (en UNIX los ficheros utilizan de*script*ores enteros).

2) *standard output* (salida estándar): normalmente asignada a la pantalla del terminal; usa el de*script*or 1.

3) *standard error* (salida estándar de errores): normalmente asignada a la pantalla del terminal; utiliza el de*script*or 2.

Esto nos indica que cualquier programa ejecutado desde el *shell* tendrá por defecto la entrada asociada al teclado del terminal, su salida hacia la pantalla y, en el caso de producirse errores, también los envía a la pantalla.

Además, los *shell*s suelen proporcionar los tres mecanismos siguientes:

1) Redirección: dado que los dispositivos de E/S y los ficheros se tratan de la misma manera en UNIX, el *shell* los trata a todos simplemente como ficheros. Desde el punto de vista del usuario, se pueden reasignar los de*script*ores de los ficheros para que los flujos de datos de un de*script*or vayan a cualquier otro de*script*or; a esto se le llama redirección. Por ejemplo, nos referiremos a la redirección de los de*script*ores 0 o 1 como a la redirección de la E/S estándar.

2) Tuberías (*pipes*): la salida estándar de un programa puede usarse como entrada estándar de otro por medio de *pipes*. Varios programas pueden ser conectados entre sí mediante *pipes* para formar lo que se denomina un *pipeline*.

3) Concurrencia de programas de usuario: los usuarios pueden ejecutar varios programas simultáneamente, indicando que su ejecución se va a producir en segundo plano (*background*), en términos opuestos a primer plano (o *foreground*), donde se tiene un control exclusivo de pantalla. Otra utilización consiste en permitir trabajos largos en segundo plano cuando interactuamos con el *shell* con otros programas en primer plano.

En los *shell*s UNIX/Linux estos tres aspectos suponen en la práctica:

- **Redirección:** un comando va a poder recibir su entrada o salida desde otros ficheros o dispositivos.

Ejemplo

veamos

```
comando op fichero
```

donde op puede ser:

  - < : recibir entrada del fichero.
  - > : enviar salida al fichero.
  - >> : indica que se añada la salida (por defecto, con > se crea de nuevo el fichero).

- *Pipes*: encadenamiento de varios comandos, con transmisión de sus datos:

  ```
  comando1 | comando2 | comando3
  ```

  Esta instrucción nos indica que comando1 recibirá entrada posiblemente de teclado, enviará su salida a comando2, que la recibirá como entrada, y éste producirá salida hacia comando3, que la recibe y produce su salida hacia salida estándar (la pantalla, por defecto).

- Concurrencia en segundo plano: cualquier comando ejecutado con el '&' al final de línea se ejecuta en segundo plano y el *prompt* del *shell* se devuelve inmediatamente mientras continúa su ejecución. Podemos seguir la ejecución de los comandos con el comando *ps* y sus opciones, que nos permite ver el estado de los procesos en el sistema. Y también disponemos de la orden *kill*, que nos permite eliminar procesos que todavía se estén ejecutando o que hayan entrado en alguna condición de error: *kill 9 pid* permite "matar" el proceso número pid. pid es el identificador asociado al proceso, un número entero que el sistema le asigna y que puede obtenerse con el comando *ps*.

4.2. *Shells* disponibles

La independencia del *shell* respecto al *kernel* del operativo (el *shell* es sólo una capa de interfaz), nos permite disponer de varios de ellos en el sistema [Qui01]. Algunos de los más comunes son:

a) El *shell* Bash (*bash*). El *shell* GNU/Linux por defecto.

b) El *shell* Bourne (sh). Éste ha sido desde siempre el *shell* estándar UNIX, y el que todos los UNIX poseen en alguna versión. Normalmente, es el *shell* por defecto del administrador (*root*). En GNU/Linux suele ser el anterior Bash, una versión mejorada del Bourne. El sh fue creado por Stephen Bourne en AT&T a finales de los setenta. El indicador (o *prompt*) por defecto suele ser un '$' (en *root* un '#').

c) El *shell* Korn (ksh). Es un superconjunto del Bourne (se mantiene cierta compatibilidad), escrito en AT&T por David Korn (a mediados de los ochenta), en el cual se hizo cierta mezcla de funcionalidades del Bourne y del C, más algún añadido. El *prompt* por defecto es el $.

d) El *shell* C (csh). Fue desarrollado en la Universidad de Berkeley por Bill Joy a finales de los setenta y tiene unos cuantos añadidos interesantes al Bourne, como un histórico de comandos, alias, aritmética desde la línea de comandos, completa nombres de ficheros y hace control de trabajos en segundo plano. El *prompt* por defecto para los usuarios es '%'. Los usuarios UNIX suelen preferir este *shell* como interactivo, pero los administradores UNIX prefieren utilizar el Bourne, ya que los *script*s suelen quedar más compactos, y la ejecución suele ser más rápida. Por otro lado, una ventaja de los *script*s en C *shell* es que, como su nombre indica, su sintaxis está basada en el lenguaje C (aunque no igual).

e) Otros, como versiones restringidas o especializadas de los anteriores.

El *shell* Bash (*Bourne again shell*) [Bas] [Coo] ha adquirido importancia desde su inclusión en los sistemas GNU/Linux como *shell* por defecto. Este *shell* forma parte del software del proyecto GNU. Es un intento de combinar los tres *shell* anteriores (Bourne, C y Korn), manteniendo la sintaxis del *shell* Bourne original. Es en el que nos vamos a fijar para desarrollar ejemplos posteriores.

Una forma rápida de conocer bajo qué *shell* nos encontramos como usuarios es mediante la variable $*SHELL*, desde una línea de comandos con la instrucción:

```
echo $SHELL
```

Algunas cuestiones que encontraremos comunes a todos los *shell*s:

- Todos permiten la escritura de *shell script*s, que son luego interpretados ejecutándolos bien por el nombre (si el fichero tiene permiso de ejecución) o bien pasándolo como parámetro al comando del *shell*.

- Los usuarios del sistema tienen un *shell* por defecto asociado a ellos. Esta información se proporciona al crear las cuentas de los usuarios. El administrador asigna un *shell* a cada usuario, o bien si no se asigna el *shell* por defecto (*bash* en GNU/Linux). Esta información se guarda en el fichero de *passwords* en */etc/passwd,* y puede cambiarse con la orden *chsh,* esta misma orden con opción -l nos lista los *shell*s disponibles en el sistema (o ver también */etc/shells*).

- Cada *shell* es en realidad un comando ejecutable, normalmente presente en los directorios /bin en GNU/Linux (o /usr/bin).

- Se pueden escribir *shell script*s en cualquiera de ellos, pero ajustándose a la sintaxis de cada uno, que es normalmente diferente (a veces hay sólo pequeñas diferencias). La sintaxis de las construcciones, así como los comandos internos, están documentados en la página man de cada *shell* (man bash por ejemplo).

- Cada *shell* tiene algunos ficheros de arranque asociados (ficheros de inicialización), cada usuario puede adaptarlos a sus necesidades, incluyendo código, variables, caminos (path)...

- La potencia en la programación está en combinar la sintaxis de cada *shell* (de sus construcciones), con los comandos internos de cada *shell*, y una serie de comandos UNIX muy utilizados en los *script*s, como por ejemplo los cut, sort, cat, more, echo, grep, wc, awk, sed, mv, ls, cp...

Nota

Para la programación shell es recomendable tener un buen conocimiento de estos comandos UNIX, y sus diferentes opciones.

- Si como usuarios estamos utilizando un *shell* determinado, nada impide arrancar una copia nueva de *shell* (lo llamamos sub*shell*), ya sea el mismo u otro diferente. Sencillamente, lo invocamos por el nombre del ejecutable, ya sea el sh, bash, csh o ksh. También cuando ejecutamos un *shell script* se lanza un sub*shell* con el *shell* que corresponda para ejecutar el *script* pedido.

Algunas diferencias básicas entre ellos [Qui01]:

a) Bash es el *shell* por defecto en GNU/Linux (si no se especifica lo contrario al crear la cuenta del usuario). En otros sistemas UNIX suele ser el *shell* Bourne (sh). Bash es compatible con sh, y además incorpora algunas características de los otros *shell*s, csh y ksh.

b) Ficheros de arranque: sh, ksh tienen *.profile* (en la cuenta del usuario, y se ejecuta en el *login* del usuario) y también ksh suele tener un *.kshrc* que se ejecuta a continuación, csh utiliza *.login* (se ejecuta al iniciarse el *login* del usuario una sola vez), *.logout* (antes de la salida de la sesión del usuario) y *.cshrc* (parecido al *.profile*, en cada sub*shell* C que se inicia). Y Bash utiliza el *.bashrc* y el *.bash_profile*. Además, el administrador puede colocar variables y caminos comunes en el fichero */etc/profile* que se ejecutará antes que los ficheros que tenga cada usuario. Los ficheros de inicialización de los *shell* se ponen en la cuenta del usuario al crearla (normalmente se copian del directorio /etc/skel), donde el administrador puede dejar unos esqueletos de los ficheros preparados.

c) Los *script*s de configuración del sistema o de servicios suelen estar escritos en *shell* Bourne (sh), ya que la mayoría de los UNIX así los utilizaban. En GNU/Linux también nos podemos encontrar con algunos en Bash, y también en otros lenguajes de *script* no asociados a los *shell* como perl o python.

d) Podemos identificar en qué *shell* se ejecuta el *script* mediante el comando *file*, por ejemplo *file <nombrescript>*. O bien examinando la primera línea del *script*, que suele ser: *#!/bin/nombre*, donde nombre es *bash, sh, csh, ksh*... Esta línea le dice, en el momento de ejecutar el *script*, qué *shell* hay que utilizar a la hora de interpretarlo (o sea, qué sub*shell* hay que lanzar para ejecutarlo). Es importante que todos los *script*s la contengan, ya que si no, se intentarán ejecutar en el *shell* por defecto (Bash en nuestro caso), y la sintaxis puede no corresponder, causando muchos errores sintácticos en la ejecución.

4.3. Variables del sistema

Algunas variables de sistema útiles (podemos verlas por ejemplo, mediante el comando *echo*), que pueden consultarse ya sea en la línea de comandos o dentro en la programación de *shell*s *script* son:

Variable	Valor Ejemplo	Descripción
HOME	/home/juan	Directorio raíz del usuario
LOGNAME	Juan	Id del usuario en el *login*
PATH	/bin:/usr/local/bin:/usr/X11/bin	Caminos
SHELL	/bin/bash	*Shell* del usuario
PS1	$	Prompt del *shell*, el usuario puede cambiarlo
MAIL	/var/mail/juan	Directorio del buzón de correo
TERM	xterm	Tipo de terminal que el usuario utiliza
PWD	/home/juan	Directorio actual del usuario

Las diferentes variables del entorno pueden verse con el comando *env*. Por ejemplo:

```
$ env
SSH_AGENT_PID = 598
MM_CHARSET = ISO-8859-15
TERM = xterm
DESKTOP_STARTUP_ID =
SHELL = /bin/bash
WINDOWID = 20975847
LC_ALL = es_ES@euro
USER = juan
LS_COLORS = no = 00:fi = 00:di = 01;34:ln = 01;
SSH_AUTH_SOCK = /tmp/ssh-wJzVY570/agent.570
SESSION_MANAGER = local/aopcjj:/tmp/.ICE-unix/570
USERNAME = juan
PATH=/soft/jdk/bin:/usr/local/bin:/usr/bin:/bin:/usr/bin/
X11:/usr/games
MAIL = /var/mail/juan
PWD = /etc/skel
JAVA_HOME = /soft/jdk
LANG = es_ES@euro
GDMSESSION = Gnome
JDK_HOME = /soft/jdk
SHLVL = 1
HOME = /home/juan
GNOME_DESKTOP_SESSION_ID = Default
LOGNAME = juan
DISPLAY = :0.0
```

```
COLORTERM = gnome-terminal
XAUTHORITY = /home/juan/.Xauthority
_ = /usr/bin/env
OLDPWD = /etc
```

4.4. Programación *scripts* en Bash

Aquí veremos algunos conceptos básicos de los *shell script*s en Bash, se recomienda ampliar en [Bas] [Coo].

Todos los *script*s Bash tienen que comenzar con la línea:

```
#!/bin/bash
```

Esta línea le indica al *shell* usado por el usuario, el que tengamos activo en el momento, con qué *shell* hay que ejecutar el *script* que aparezca a continuación.

El *script* puede ejecutarse de dos modos diferentes:

1) Por ejecución directa desde la línea de comandos, siempre que tenga permiso de ejecución. Si no se da el caso, ponemos el permiso con: *chmod +x script*.

2) Por ejecución mediante el *shell*, llamamos al *shell* explícitamente: */bin/bash script*.

Hay que tener en cuenta que, sea cuál sea el método de ejecución, siempre estamos creando un sub*shell* donde se va a ejecutar nuestro *script*.

4.4.1. Variables en Bash

La asignación de variables se realiza por:

```
variable = valor
```

El valor de la variable se puede ver con:

```
echo $variable
```

donde '$' nos hace referencia al valor de la variable.

La variable por defecto sólo es visible en el *script* (o en el *shell*). Si la variable tiene que ser visible fuera del *script*, a nivel de *shell* o de cualquier *shell* hijo (o

subshell) que se genere a posteriori, será necesario "exportarla" además de asignarla. Podemos hacer dos cosas:

- Asignar y exportar después:

```
var = valor
export var
```

- Exportar en la asignación:

```
export var = valor
```

En los *script*s Bash tenemos algunas variables predeterminadas accesibles:

- $1-$N: Guarda los argumentos pasados como parámetros al *script* desde la línea de comandos.
- $0: Guarda el nombre del *script*, sería el parámetro 0 de la línea de comandos.
- $*: Guarda todos los parámetros del 1 al N en esta variable.
- $: Guarda todos los parámetros, pero con comillas dobles (" ") en cada uno de ellos.
- $?: "Status": guarda el valor devuelto por el último comando ejecutado. Útil para verificar condiciones de error, ya que UNIX suele devolver 0 si la ejecución ha sido correcta, y un valor diferente como código de error.

Otra cuestión importante en las asignaciones es el uso de las comillas:

- Las "dobles" permiten que sea considerado todo como una unidad.
- Las 'simples' son parecidas, pero se ignoran los caracteres especiales que se encuentren dentro.
- Las inclinadas hacia la izquierda `comando`, son utilizadas para evaluar el interior, si hay alguna ejecución o sustitución que hacer. Primero se ejecuta el contenido, y se sustituye lo que había por el resultado de la ejecución. Por ejemplo: var = `ls` guarda el listado del directorio en $var.

4.4.2. Comparaciones

Para las condiciones se suele utilizar la orden *test expresión* o directamente *[expresión]*. Podemos agrupar las condiciones disponibles en:

- Comparación numérica: -eq, -ge, -gt, -le, -lt, -ne, correspondiendo a: igual que, más grande o igual que (ge), más grande que, menor o igual que (le), menor que, distinto que.

- Comparación de cadenas: :=, !=, -n, -z, correspondiendo a cadenas de caracteres: iguales, diferentes, con longitud mayor que 0, longitud igual a cero o vacío.

- Comparación de ficheros: -d, -f -r, -s, -w, -x. El fichero es: un directorio, un fichero ordinario, es leíble, es no vacío, es escribible, es ejecutable.

- Booleanos entre expresiones: !, -a, -o, condiciones de not, and y or.

4.4.3. Estructuras de control

Respecto a la programación interna del *script*, hay que pensar que nos vamos a encontrar básicamente con:

- Comandos propios del operativo.
- Comandos propios internos del *shell* Bash (ver: man bash).
- Las estructuras de control propias de programación (for, while...), con la sintaxis propia de Bash.

La sintaxis básica de las estructuras de control es la siguiente:

a) Estructura *if...then*, se evalúa la expresión y si se obtiene un valor cierto, entonces se ejecutan los commands.

```
if [ expresion ]
  then
    commands
fi
```

b) Estructura if..then...else, se evalúa la expresión, y si se obtiene un valor de cierto, entonces se ejecutan los commands1, en caso contrario se ejecutan los comands2:

```
if [ expresion ]
  then
    commands1
  else
    commands2
fi
```

c) Estructura if..then...else if...else, misma utilización que la anterior, con anidamientos de estructuras if.

```
if [ expresion ]
  then
    commands
  elif [ expresion2 ]
  then
    commands
```

```
  else
    commands
fi
```

d) Estructura *case select,* estructura de selección múltiple según valor de selección (en *case*)

```
case string1 in
  str1)
    commands;;
  str2)
    commands;;
  *)
    commands;;
esac
```

Nota

Los shells como Bash, ofrecen un conjunto amplio de estructuras de control que los hace comparables a cualquier otro lenguaje.

e) Bucle *for*, sustitución de variable por cada elemento de la lista:

```
for var1 in list
do
    commands
done
```

f) Bucle *while*, mientras se cumpla la expresión:

```
while [ expresion ]
do
    commands
done
```

g) Bucle *until*, hasta que se cumpla la expresión:

```
until [ expresion ]
do
    commands
done
```

h) Declaración de funciones:

```
fname() {
    commands
}
```

o bien con llamada acompañada de parámetros:

```
fname2(arg1,arg2...argN) {
    commands
}
```

y la llamadas de la función con fname o fname2 p1 p2 p3 ... pN.

5. Herramientas de gestión de paquetes

En cualquier distribución, los paquetes son el elemento básico para tratar las tareas de instalación de nuevo software, actualización del existente o eliminación del no utilizado.

Básicamente, un paquete es un conjunto de ficheros que forman una aplicación o una unión de varias aplicaciones relacionadas, normalmente formando un único fichero (denominado paquete), con un formato propio y normalmente comprimido, que es el que se distribuye, ya sea vía CD, disquete o mediante acceso a servicios de ftp o web.

El uso de paquetes facilita añadir o quitar software al considerarlo una unidad y no tener que trabajar con los ficheros individuales.

En el contenido de la distribución (sus CD) los paquetes suelen estar agrupados por categorías como: a) base: paquetes indispensables para el funcionamiento del sistema (útiles, programas de inicio, bibliotecas de sistema); b) sistema: útiles de administración, comandos de utilidad; c) desarrollo (*developement*): útiles de programación: editores, compiladores, depuradores... d) gráficos: controladores e interfaces gráficas, escritorios, gestores de ventanas... e) otras categorías.

Normalmente, para la instalación de un paquete será necesario efectuar una serie de pasos:

1) Previo (preinstalación): comprobar que existe el software necesario (y con las versiones correctas) para su funcionamiento (dependencias), ya sean bibliotecas de sistema u otras aplicaciones que sean usadas por el software.

2) Descomprimir el contenido del paquete, copiando los ficheros a sus localizaciones definitivas, ya sean absolutas (tendrán una posición fija) o si se permite reubicarlas a otros directorios.

3) Postinstalación: retocar los ficheros necesarios, configurar posibles parámetros del software, adecuarlo al sistema...

Dependiendo de los tipos de paquetes, estos pasos pueden ser automáticos en su mayoría (así es en el caso de RPM [Bai03] y DEB [Deb02]), o pueden necesitar hacerlos todos a mano (caso .tgz) dependiendo de las herramientas que proporcione la distribución.

Veremos a continuación quizás los tres paquetes más clásicos de la mayoría de las distribuciones. Cada distribución tiene uno por estándar y soporta alguno de los demás.

5.1. Paquete TGZ

Los paquetes TGZ son quizás los de utilización más antigua. Las primeras distribuciones de GNU/Linux los utilizaban para instalar el software, y aún varias distribuciones los usan (por ejemplo, Slackware) y algunos UNIX comerciales. Son una combinación de ficheros unidos por el comando tar en un único fichero .tar, que luego ha sido comprimido por la utilidad gzip, suele aparecer con la extensión .tgz o bien .tar.gz. Asimismo, hoy en día es común encontrar los tar.bz2 que utilizan en lugar de gzip otra utilidad llamada bzip2, que en algunos casos consigue mayor compresión del archivo.

Nota

Los paquetes TGZ son una herramienta básica a la hora de instalar software no organizado. Además, son una herramienta útil para realizar procesos de backup y restauración de archivos.

Este tipo de paquete no contiene ninguna información de dependencias, y puede presentar tanto contenido de aplicaciones en formato binario como en código fuente. Podemos considerarlo como una especie de colección de ficheros comprimida.

En contra de lo que pudiera parecer, es un formato muy utilizado, y sobre todo por creadores o distribuidores de software externo a la distribución. Muchos creadores de software que trabajan para plataformas varias, como varios UNIX comerciales, y diferentes distribuciones de GNU/Linux lo prefieren como sistema más sencillo y portable.

Un ejemplo de este caso es el proyecto GNU, que distribuye su software en este formato (en forma de código fuente), ya que puede utilizarse en cualquier UNIX, ya sea un sistema propietario, una variante BSD o una distribución GNU/Linux.

Si se trata de formato binario, tendremos que tener en cuenta que sea adecuado para nuestro sistema, por ejemplo, suele ser común alguna denominación como la que sigue (en este caso, la versión 1.4 del navegador web Mozilla):

```
mozilla-i686-pc-linux-gnu-1.4-installer.tar.gz
```

donde tenemos el nombre del paquete, como Mozilla, arquitectura a la que ha destinado i686 (Pentium II o superiores o compatibles), podría ser i386, i586, i686, k6 (amd k6), k7 (amd athlon), amd64 u x86_64 (para AMD64 y algunos intel de 64bits con em64t), o ia64 (intel Itaniums) otras para arquitecturas de otras máquinas como sparc, powerpc, mips, hppa, alpha... después nos indica qué es para Linux, en una máquina PC, la versión del software 1.4.

Si fuese en formato fuente, suele aparecer como:

```
mozilla-source-1.4.tar.gz
```

donde se nos indica la palabra *source*; en este caso no menciona la versión de arquitectura de máquina, esto nos indica que está preparado para compilarse en diferentes arquitecturas.

De otro modo, habría diferentes códigos para cada sistema operativo o fuente: GNU/Linux, Solaris, Irix, bsd...

El proceso básico con estos paquetes consiste en:

1) Descomprimir el paquete (no suelen utilizar *path* absoluto, con lo que se pueden descomprimir en cualquier parte):

```
tar -zxvf fichero.tar.gz (o fichero.tgz)
```

Con el comando *tar* ponemos opciones de z: descomprimir, x: extraer ficheros, v: ver proceso, f: dar nombre del fichero a tratar.

También se puede hacer por separado (sin la z del tar):

```
gunzip fichero.tar.gz
```

(nos deja un fichero tar)

```
tar -xvf fichero.tar
```

2) Una vez tenemos descomprimido el tgz, tendremos los ficheros que contenía, normalmente el software debe incluir algún fichero de tipo *readme* o *install*, donde nos especificarán las opciones de instalación paso a paso, y también posibles dependencias del software.

En primer lugar habrá que verificar las dependencias por si disponemos del software adecuado, y si no, buscarlo e instalarlo.

Si se trata de un paquete binario, la instalación suele ser bastante fácil, ya que o bien directamente ya será ejecutable donde lo hayamos dejado, o traerá algún instalador propio. Otra posibilidad será que tengamos que hacerlo manualmente, con lo que bastará con copiar (cp -r, copia recursiva) o mover (comando *mv*) el directorio a la posición deseada.

Otro caso es el formato de código fuente. Entonces, antes de instalar el software tendremos que pasar por una compilación. Para eso habrá que leerse con cierto detalle las instrucciones que lleve el programa. Pero la mayoría de desa-

rrolladores usan un sistema de GNU llamado autoconf (de autoconfiguración), en el que habitualmente se usan los siguientes pasos (si no aparecen errores):

- *./configure*: se trata de un *script* que configura el código para poder ser compilado en nuestra máquina, verifica que existan las herramientas adecuadas. La opción --prefix = directorio permite especificar dónde se instalará el software.

- *make*: compilación propiamente dicha.

- *make install*: instalación del software a un lugar adecuado, normalmente especificado previamente como opción al configure o asumida por defecto.

Éste es un proceso general, pero depende del software que lo siga o no, hay casos bastante peores donde todo el proceso se tiene que realizar a mano, retocando ficheros de configuración, o el mismo *makefile*, y/o compilando uno a uno los ficheros, pero esto, por suerte, es cada vez menos habitual.

En caso de querer borrar el software instalado, habrá que utilizar el desinstalador si nos lo proporcionan, o si no, borrar directamente el directorio o ficheros que se instalaron, teniendo cuidado de posibles dependencias.

Los paquetes tgz son bastante habituales como mecanismo de *backup* en tareas de administración, por ejemplo, para guardar copias de datos importantes, hacer *backups* de cuentas de usuario, o guardar copias antiguas de datos que no sabemos si volveremos a necesitar. Suele utilizarse el siguiente proceso: supongamos que queremos guardar copia del directorio “dir” tar -cvf dir.tar dir (c: compactar dir en el fichero dir.tar) gzip dir.tar (comprimir) o bien en una sola instrucción como:

```
tar -zcvf dir.tgz dir
```

El resultado será un fichero dir.tgz. Hay que ser cuidadoso si nos interesa conservar los atributos de los ficheros, y permisos de usuario, así como posiblemente ficheros de enlace (*links*) que pudieran existir (deberemos examinar las opciones de tar para que se ajuste a las opciones de *backup* deseadas).

5.2. Fedora/Red Hat: paquetes RPM

El sistema de paquetes RPM [Bai03] creado por Red Hat supone un paso adelante, ya que incluye la gestión de dependencias y tareas de configuración del software. Además, el sistema guarda una pequeña base de datos con los paquetes ya instalados, que puede consultarse y se actualiza con las nuevas instalaciones.

Los paquetes RPM, por convención, suelen usar un nombre como:

```
paquete-version-rev.arq.rpm
```

donde *paquete* es el nombre del software, *version* es la numeración de versión del software, *rev* suele ser la revisión del paquete RPM, que indica las veces que se ha construido, y *arq,* la arquitectura a la que va destinado el paquete, ya sea Intel/AMD (i386, i586, i686, x86_64, em64t, ia64) u otras como alpha, aparc, ppc... La "arquitectura" Noarch suele usarse cuando es independiente, por ejemplo, un conjunto de *script*s, y src en el caso de que se trate de paquetes de código fuente. La ejecución típica incluye la ejecución de rpm, las opciones de la operación a realizar, junto con uno o más nombres de paquetes por procesar juntos.

Nota

El paquete: apache-1.3.19-23.i686.rpm indicaría que se trata del software Apache (el servidor web), en su versión 1.3.19, revisión del paquete RPM 23, para arquitecturas Pentium II o superiores.

Las operaciones típicas con los paquetes RPM incluyen:

- **Información del paquete**: se consulta sobre el paquete una información determinada, se usa la opción -q acompañada del nombre del paquete (con -p si se hace sobre un archivo rpm). Si el paquete no ha sido instalado todavía, la opción sería -q acompañada de la opción de información que se quiera pedir, y si se quiere preguntar a todos los paquetes instalados a la vez, la opción sería -qa. Por ejemplo, preguntas a un paquete instalado:

Consulta	Opciones RPM	Resultados
Archivos	`rpm -ql`	Lista de los archivos que contiene
Información	`rpm -qi`	Descripción del paquete
Requisitos	`rpm -qR`	Requisitos previos, bibliotecas o software

- **Instalación:** simplemente rpm -i paquete.rpm, o bien con URL donde encontrar el paquete, para descargarlo desde servidores FTP o web, sólo hay que utilizar la sintaxis ftp:// o http:// para dar la localización del paquete.

  La instalación podrá realizarse siempre que se estén cumpliendo las dependencias del paquete, ya sea software previo o bibliotecas que deberían estar instaladas. En caso de no cumplirlo, se nos listará qué software falta, y el nombre del paquete que lo proporciona. Puede forzarse la instalación (a riesgo de que no funcione) con las opciones `--force` o `--nodeps`, o simplemente se ignora la información de las dependencias.

  La tarea de instalación (realizada por rpm) de un paquete conlleva diferentes subtareas: a) verificar las posibles dependencias; b) examinar por conflictos con otros paquetes previamente instalados; c) efectuar tareas previas a la instalación; c) decidir qué hacer con los ficheros de configuración asociados al paquete si previamente existían; d) desempaquetar los ficheros y colocarlos en el sitio correcto; e) llevar a cabo tareas de postinstalación; finalmente, f) almacenar registro de las tareas efectuadas en la base de datos de RPM.

- **Actualización:** equivalente a la instalación pero comprobando primero que el software ya existe rpm -U paquete.rpm. Se encargará de borrar la instalación previa.

- **Verificación:** durante el funcionamiento normal del sistema, muchos de los archivos instalados cambian. En este sentido, RPM permite verificar los archivos para detectar las modificaciones, bien por proceso normal, bien por algún error que podría indicar datos corrompidos. Mediante rpm -V paquete verificamos un paquete concreto, y mediante rpm -Va los verificará todos.

- **Eliminación:** borrar el paquete del sistema RPM (-e o --erase); si hay dependencias, puede ser necesario eliminar otros previamente.

  Ejemplo

  Para un caso remoto:

  rpm -i ftp://sitio/directorio/paquete.rpm

  nos permitiría descargar el paquete desde el sitio ftp o web proporcionado, con su localización de directorios, y proceder en este caso a la instalación del paquete.

Hay que vigilar con la procedencia de los paquetes, y sólo utilizar fuentes de paquetes conocidas y fiables, ya sea del propio fabricante de la distribución, o sitios en los que confiemos. Normalmente se nos ofrece, junto con los paquetes, alguna "firma" digital de éstos para que podamos comprobar su autenticidad. Suelen utilizarse las sumas md5 para comprobar que el paquete no se ha alterado, y otros sistemas como GPG (versión Gnu de PGP) para comprobar la autenticidad del emisor del paquete. Asimismo, podemos encontrar en Internet diferentes almacenes de paquetes RPM, donde están disponibles para diferentes distribuciones que usen o permitan el formato RPM.

Nota

Ver el sitio: www.rpmfind.net.

Para un uso seguro de paquetes, actualmente los repositorios (oficiales, y algunos de terceros) firman electrónicamente los paquetes, por ejemplo con el mencionado GPG; esto nos permite asegurar (si disponemos de las firmas) que los paquetes proceden de la fuente fiable. Normalmente, cada proveedor (el repositorio) incluye unos ficheros de firma PGP con la clave para su sitio. De los repositorios oficiales normalmente ya vienen instaladas, si proceden de terceros, deberemos obtener el fichero de clave, e incluirla en RPM, típicamente:

```
$ rpm –import GPG-KEY-FILE
```

Siendo GPP-KEY-FILE el fichero clave GPG o la URL de dicho fichero, normalmente este fichero también tendrá suma md5 para comprobar su integridad. Y podemos conocer las claves existentes en el sistema con:

```
$ rpm -qa | grep ^gpg-pubkey
```

podemos observar más detalles a partir de la clave obtenida:

```
$ rpm -qi gpg-key-xxxxx-yyyyy
```

Para un paquete rpm concreto podremos comprobar si dispone de firma y cuál se ha utilizado con:

```
$ rpm –checksig -v <paquete>.rpm
```

Y para verificar que un paquete es correcto en base a las firmas disponibles, puede comprobarse con:

```
$ rpm -K <paquete.rpm>
```

Debemos ser cuidadosos en importar sólo aquellas claves de los sitios en los que confiemos. Cuando RPM encuentre paquetes con firma que no tengamos en nuestro sistema, o el paquete no esté firmado, nos avisará, y la acción ya dependerá de nuestra actuación.

En cuanto al soporte RPM en las distribuciones, en Fedora (Red Hat, y también en sus derivadas), RPM es el formato por defecto de paquetes y el que usa ampliamente la distribución para las actualizaciones, y la instalación de software. En Debian se utiliza el formato denominado DEB (como veremos), hay soporte para RPM (existe el comando rpm), pero sólo para consulta o información de paquetes. En el caso de que sea imprescindible instalar un paquete rpm en Debian, se recomienda utilizar la utilidad *alien*, que permite convertir formatos de paquetes, en este caso de RPM a DEB, y proceder a la instalación con el paquete convertido.

Además del sistema base de empaquetado de la distribución, hoy en día cada una suele soportar un sistema de gestión de software intermedio de más alto nivel, que añade una capa superior al sistema base, facilitando las tareas de gestión del software, y añadiendo una serie de utilidades para controlar mejor el proceso.

En el caso de Fedora (Red Hat y derivados) se utiliza el sistema YUM, que permite como herramienta de más alto nivel la instalación y gestión de paquetes en sistemas rpm, así como la gestión de automática de dependencias entre los paquetes. Permite acceder a múltiples repositorios diferentes, centraliza su configuración en un fichero (/etc/yum.conf habitualmente), y tiene una interfaz de comandos simple.

Nota

YUM en: http://linux.duke.edu/projects/yum

La configuración de yum se basa en:

/etc/yum.config (fichero de opciones)

/etc/yum (directorio para algunas utilidades asociadas)

/etc/yum.repos.d (directorio de especificación de repositorios, un fichero para cada uno, se incluye información del acceso, y localización de las firmas gpg).

Para las operaciones típicas de yum un resumen sería:

Orden	Descripción
yum install <nombre>	Instalar el paquete con el nombre
yum update <nombre>	Actualizar un paquete existente
yum remove <nombre>	Eliminar paquete
yum list <nombre>	Buscar paquete por nombre (sólo nombre)
yum search <nombre>	Buscar más ampliamente
yum provices <file>	Buscar paquetes que proporcionen el fichero
yum update	Actualizar todo el sistema
yum upgrade	Ídem al anterior incluyendo paquetes adicionales

Para finalizar, Fedora también ofrece un par de utilidades gráficas para YUM, pup para controlar las actualizaciones recientes disponibles, y pirut como paquete de gestión de software. También existen algunas otras como yumex*,* con mayor control de la configuración interna de yum.

5.3. Debian: paquetes DEB

Debian tiene herramientas interactivas como tasksel, que permiten escoger unos subconjuntos de paquetes agrupados por tipo de tareas: paquetes para X, para desarrollo, para documentación, etc., o como dselect, que nos permite navegar por toda la lista de paquetes disponible (hay miles), y escoger aquellos que queramos instalar o desinstalar. De hecho estas son sólo un front-end del gestor de software nivel intermedio APT.

En el nivel de línea de comandos dispone de dpkg, que es el comando de más bajo nivel (*base*, sería el equivalente a rpm), para gestionar directamente los paquetes DEB de software [Deb02], típicamente dpkg -i paquete.deb para realizar la instalación. Pueden realizarse todo tipo de tareas, de información, instalación, borrado o cambios internos a los paquetes de software.

El nivel intermedio (como el caso de Yum en Fedora) lo presentan las herramientas APT (la mayoría son comandos *apt-xxx*). APT permite gestionar los paquetes a través de una lista de paquetes actuales y disponibles a partir de varias fuentes de software, ya sea desde los propios CD de la instalación, sitios ftp o web (HTTP). Esta gestión se hace de forma transparente, de manera que el sistema es independiente de las fuentes de software.

La configuración del sistema APT se efectúa desde los archivos disponibles en /etc/apt, donde /etc/apt/sources.list es la lista de fuentes disponibles; un ejemplo podría ser:

```
deb http://http.us.debian.org/debian stable main contrib non-free
debsrc http://http.us.debian.org/debian stable main contrib non-free
deb http://security.debian.org stable/updates main contrib non-free

#Sources Oficiales de Debian STABLE "etch"
deb http://ftp.debian.org/debian/ etch main non-free contrib
debsrc http://ftp.debian.org/debian/ etch main non-free contrib
```

Donde hay recopiladas varias de las fuentes "oficiales" para una Debian (*etch* en este caso, suponiendo ésta como *stable*), desde donde se pueden obtener los paquetes de software, así como las actualizaciones que estén disponibles. Básicamente, se especifica el tipo de fuente (web/ftp en este caso), el sitio, la versión de la distribución (stable o etch en este ejemplo), y categorías del software que se buscará (libre, o contribuciones de terceros o de licencia no libre o comercial).

Los paquetes de software están disponibles para las diferentes versiones de la distribución Debian, existen paquetes para las versiones stable, testing, y unstable. El uso de unos u otros determina el tipo de distribución (previo cambio de las fuentes de repositorios en sources.list). Pueden tenerse fuentes de paquetes mezcladas, pero no es muy recomendable, ya que se podrían dar conflictos entre las versiones de las diferentes distribuciones.

Nota

Los paquetes DEB de Debian es quizás el sistema de instalación más potente existente en GNU/Linux. Una prestación destacable es la independencia del sistema de las fuentes de los paquetes (mediante APT).

Una vez tenemos las fuentes de software configuradas, la principal herramienta para manejarlas en nuestro sistema es apt-get, que nos permite instalar, actualizar o borrar desde el paquete individual, hasta actualizar la distribución entera. Existe también un front-end a apt-get, llamado *aptitude,* cuya interfaz de opciones es prácticamente igual (de hecho podría calificarse de emulador de apt-get, ya que la interfaz es equivalente); como ventaja aporta una mejor gestión de dependencias de los paquetes y permite una interfaz interactiva. De hecho se espera que aptitude sea la interfaz por defecto en línea de comandos para la gestión de paquetes.

Algunas funciones básicas de apt-get:

- Instalación de un paquete particular:

```
apt-get install paquete
```

- Borrado de un paquete:

```
apt-get remove paquete
```

- Actualización de la lista de paquetes disponibles:

```
apt-get update
```

- Actualización de la distribución, podríamos efectuar los pasos combinados:

```
apt-get update
apt-get upgrade
apt-get dist-upgrade
```

Mediante este último proceso, podemos mantener nuestra distribución actualizada permanentemente, actualizando los paquetes instalados y verificando las dependencias con los nuevos. Unas herramientas útiles para construir esta lista es `apt-spy`, que intenta buscar los sitios oficiales más rápidos, o netselect, que nos permite probar una lista de sitios. Por otro lado, podemos buscar las fuentes oficiales (podemos configurarlas con apt-setup), o bien copiar algún fichero de fuentes disponible. Software adicional (de terceros) puede necesitar añadir otras fuentes más (a sources.list); se pueden obtener listas de sitios de fuentes disponibles (por ejemplo: http://www.apt-get.org).

La actualización del sistema en particular genera una descarga de un gran número de paquetes (en especial en `unstable`), lo que hace recomendable vaciar la caché, el repositorio local, con los paquetes descargados (se mantienen en /var/cache/apt/archive) que ya no vayan a ser utilizados, bien con `apt-get clean`, para eliminarlos todos, o bien con `apt-get autoclean` para eliminar aquellos paquetes no necesarios porque ya hay nuevas versiones y ya no serán necesarios (en principio). Hay que tener en cuenta si vamos a volver a necesitar estos paquetes por razones de reinstalación, ya que, si es así, tendremos que volver a descargarlos.

El sistema APT también permite lo que se denomina SecureAPT, que es la gestión segura de paquetes mediante verificación de sumas (md5) y la firma de fuentes de paquetes (de tipo GPG). Si durante la descarga no están disponibles las firmas, apt-get informa de ello, y genera un listado con los paquetes no firmados, pidiendo si se van a dejar instalar o no, dejando la decisión al administrador. Se obtiene la lista de fuentes confiables actuales con:

```
# apt-key list
```

Las claves gpg de los sitios oficiales de Debian son distribuidas mediante un paquete, las instalamos de este modo:

```
# apt-get install debian-archive-keyring
```

evidentemente, considerando que tenemos sources.list con los sitios oficiales. Se espera que por defecto (dependiendo de la versión Debian) estas claves ya

se instalen por defecto al iniciar el sistema. Para otros sitios no oficiales (que no proporcionen la clave en paquete), pero que consideremos confiables, podemos importar su clave, obteniéndola desde el repositorio (tendremos que consultar dónde tienen la clave disponible, no hay un estándar definido, aunque suele estar en la página web inicial del repositorio). Utilizándose apt-key add con el fichero, para añadir la clave, o también:

```
# gpg -import fichero.key
# gpg -export -armor XXXXXXXX | apt-key add -
```

siendo X un hexadecimal relacionado con la clave (ver instrucciones del repositorio para comprobar la forma recomendada de importar la clave y los datos necesarios).

Otra funcionalidad importante del sistema AP son las funciones de consulta de información de los paquetes, con la herramienta apt-cache, que nos permite interactuar con las listas de paquetes de software Debian.

Ejemplo

La herramienta apt-cache dispone de comandos que nos permiten buscar información sobre los paquetes, como por ejemplo:

- Buscar paquetes sobre la base de un nombre incompleto:

```
apt-cache search nombre
```

- Mostrar la descripción del paquete:

```
apt-cache show paquete
```

- De qué paquetes depende:

```
apt-cache depends paquete
```

Otras herramientas o funcionalidades de apt interesantes:

- apt-show-versions: nos especifica qué paquetes pueden ser actualizados (y por qué versiones, ver opción -u).

Otras tareas más específicas necesitarán realizarse con la herramienta de más bajo nivel, como dpkg. Por ejemplo, obtener la lista de archivos de un paquete determinado ya instalado:

```
dpkg -L paquete
```

La lista de paquetes entera con

```
dpkg -l
```

O buscar de qué paquete proviene un elemento (fichero por ejemplo):

```
dpkg -S fichero
```

Éste en particular funciona para paquetes instalados, `apt-file` permite también buscar para paquetes todavía no instalados.

Por último, cabe mencionar también algunas herramientas gráficas para Apt como synaptic, gnome-apt para gnome, y kpackage o adept para KDE. O las textuales ya mencionadas como aptitude o dselect.

Conclusión, cabe destacar que el sistema de gestión APT (en combinación con el base dpkg) es muy flexible y potente a la hora de gestionar las actualizaciones, y es el sistema de gestión de paquetes que se usa en Debian y sus distribuciones derivadas como Ubuntu, Kubuntu, Knoppix, Linex, etc.

6. Herramientas genéricas de administración

En el campo de la administración, también podríamos considerar algunas herramientas, como las pensadas de forma genérica para la administración. Aunque cabe destacar que para estas herramientas es difícil mantenerse al día, debido a los actuales planes de versiones de las distribuciones, que tienen una evolución muy rápida. Dejamos algunos ejemplos (pero en un momento determinado pueden no ser funcionales al completo):

a) Linuxconf: es una herramienta genérica de administración que agrupa diferentes aspectos en una interfaz de menús textual, que en las ultimas versiones evolucionó a soporte web; puede utilizarse en prácticamente cualquier distribución GNU/Linux, y soporta diversos detalles propios de cada una (por desgracia lleva tiempo sin una actualización).

Nota

Podemos encontralas en: Linuxconf http://www.solucorp.qc.c a/linuxconf

b) Webmin: es otra herramienta de administración pensada desde una interfaz web; funciona con una serie de *plugins* que se pueden añadir para cada servicio que hay que administrar; normalmente cuenta con formularios donde se especifican los parámetros de configuración de los servicios; además, ofrece la posibilidad (si se activa) de permitir administración remota desde cualquier máquina con navegador.

c) Otras en desarrollo como cPanel, ISPConfig.

Por otra parte, en los entornos de escritorio de Gnome y KDE suelen disponer del concepto de “Panel de control”, que permite gestionar tanto el aspecto visual de las interfaces gráficas, como tratar algunos parámetros de los dispositivos del sistema.

En cuanto a las herramientas gráficas individuales de administración, la propia distribución de GNU/Linux ofrece algunas directamente (herramientas que acompañan tanto a Gnome como KDE), herramientas dedicadas a gestionar un dispositivo (impresoras, sonido, tarjeta de red, etc.), y otras para la ejecución de tareas concretas (conexión a Internet, configurar arranque de servicios del sistema, configurar X Window, visualizar logs...). Muchas de ellas son simples frontends (o carátulas) a las herramientas básicas de sistema, o bien están adaptadas a particularidades de la distribución.

Cabe destacar, en especial en este apartado, a la distribución Fedora (Red Hat y derivados), que intenta disponer de diversas utilidades (más o menos minimalistas) para diferentes funciones de administración, las podemos encontrar en el escritorio (en el menú de administración), o en comandos como system-config-xxxxx para diferentes funcionalidades como gestión de: pantalla, im-

presora, red, seguridad, usuarios, paquetes, etc. Podemos ver en la figura algunas de ellas:

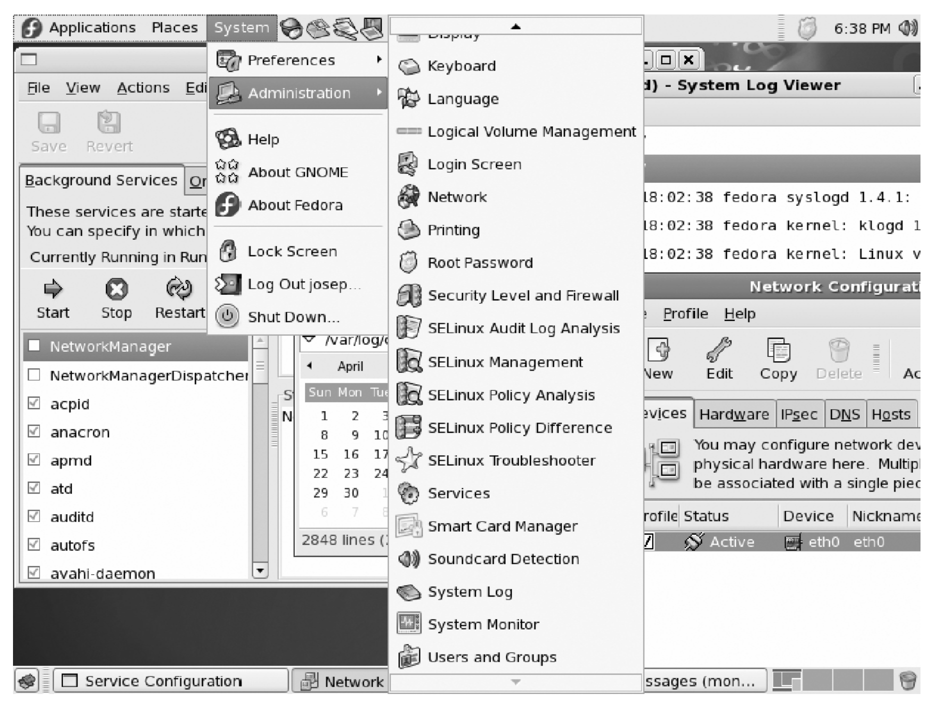


Figura 3. Varias de las utilidades gráficas de administración en Fedora

7. Otras herramientas

En el espacio limitado de esta unidad no pueden llegar a comentarse todas aquellas herramientas que nos pueden aportar beneficios para la administración. Citaremos algunas de las herramientas que podríamos considerar básicas:

- Los múltiples comandos UNIX básicos: grep, awk, sed, find, diff, gzip, bzip2, cut, sort, df, du, cat, more, file, which...

- Los editores, imprescindibles para cualquier tarea de edición cuentan con editores como: Vi, muy utilizado en tareas de administración por la rapidez de efectuar pequeños cambios en los ficheros. Vim es el editor compatible Vi, que suele traer GNU/Linux; permite sintaxis coloreada en varios lenguajes. Emacs, editor muy completo, adaptado a diferentes lenguajes de programación (sintaxis y modos de edición); dispone de un entorno muy completo y de una versión X denominada Xemacs. Joe, editor compatible con Wordstar. Y muchos otros...

Nota

Ver material asociado a curso de introducción a GNU/ Linux, o las páginas man de los comandos, o una referencia de herramientas como [Stu01].

- Lenguajes de tipo *script*, útiles para administración, como: Perl, muy útil para tratamiento de expresiones regulares, y análisis de ficheros (filtrado, ordenación, etc.). PHP, lenguaje muy utilizado en entornos web. Python, otro lenguaje que permite hacer prototipos rápidos de aplicaciones...

- Herramientas de compilación y depuración de lenguajes de alto nivel: GNU `gcc` (compilador de C y C++), `gdb` (depurador), `xxgdb` (interfaz X para gdb), `ddd` (depurador para varios lenguajes).

Actividades

1) Hacer una lectura rápida del estándar FHS, que nos servirá para tener una buena guía a la hora de buscar archivos por nuestra distribución.

2) Para repasar y ampliar conceptos y programación de *shell script*s en bash, ver: [Bas] [Coo].

3) Para los paquetes RPM, ¿cómo haríamos algunas de las siguientes tareas?:

- Conocer qué paquete instaló un determinado comando.
- Obtener la descripción del paquete que instaló un comando.
- Borrar un paquete cuyo nombre completo no conocemos.
- Mostrar todos los archivos que estaban en el mismo paquete que un determinado archivo.

4) Efectuar las mismas tareas que en la actividad anterior, pero para paquetes Debian, usando herramientas APT.

5) Actualizar una distribución Debian (o Fedora).

6) Instalar en nuestra distribución alguna herramienta genérica de administración, ya sea por ejemplo Linuxconf o Webadmin. ¿Qué nos ofrecen? ¿Entendemos las tareas ejecutadas y los efectos que provocan?

Otras fuentes de referencia e información

[Bas][Coo] ofrecen una amplia introducción (y conceptos avanzados) a la programación de *shell script*s en bash, así como numerosos ejemplos. [Qui01] comenta los diferentes *shell*s de programación en GNU/Linux, así como sus semejanzas y diferencias.

[Deb02][Bai03] ofrecen una amplia visión de los sistemas de paquetes de software de las distribuciones Debian y Fedora/Red Hat.

[Stu] es una amplia introducción a las herramientas disponibles en GNU/Linux.